More Classes and Objects

Equality, inner classes, packages, static
Equal Objects

- Suppose I have
  - Person susan = new Person();
  - Person sue = new Person();

- They might describe the same person.
  - How can I tell if they are the same?
  - All the same attributes!
    - same name, age, weight, etc.
Testing Equal Objects

How can we test?

- Compare each individual attribute.
  
  ```java
  if( sue.name.equals(susan.name) &&
      sue.age == susan.age &&
      sue.weight == susan.weight && ... )
  ```

Better way…

- Put this test in a method called “equals”.
  
  - “equals” method is standard in objects.
  - In fact always there, even if you don’t specify it!
    - We’ll see why after inheritance.

  ```java
  if( sue.equals(susan) )
  ```

Remember: Strings have to be compared with “.equals” not “==”.
public class Person {
    public int age = 0;
    public String name = "";
    public int weight = 0;

    public boolean equals(Person p) {
        return (this.age == p.age) &&
                (this.weight == p.weight) &&
                this.name.equals(p.name);
    }
}

How’s this work?

“this” isn’t necessary, but helps avoid confusion.

Your turn: Use “equals” in an example!
Inner classes

Class defined inside another class.

Whoa, why do this?

Reasons
1. Sometimes convenient – define right where you need it and nowhere else.
2. Hidden from other classes – only defined for the thing that needs it.

– And the Big Reason (for now) –

3. Handy for returning multiple things from a method – and would never need this class anywhere else!
Inner Class Format

public class OuterClass
{
    ...

    private class InnerClass
    {
        ...
    }
}

Why private? Well if it was public, then anyone could access it. In that case, should just make it a separate class.
Example That Needs Inner Class

- **Want a method that returns both min and max value of an array.**

```java
public class SecretCodebreaker {
    ...
    various methods...

    //return min and max of array
    private ??? getMinAndMax(int[] numbers) {
        int min = numbers[0];
        int max = numbers[0];
        for(int i=0; i< numbers.length; i++) {
            if(numbers[i] < min) min = numbers[i];
            if(numbers[i] > max) max = numbers[i];
        }
        return min???, max????;
    }
}
```

- **Problem:** Can’t return both min and max.

- **Solutions:**
  1. **Put in double array.**
     - problem: awkward!
  2. **Make instance variables.**
     - problem: poor form. Min and max may not be an appropriate attribute (state) of the class.
  3. **Make new inner class that holds both.**
     - notice the method is private, so not used in other classes.
public class SecretCodebreaker
{
    …various methods…

    //return min and max of array
    private Duo getMinAndMax(int[] numbers)
    {
        int min = numbers[0];
        int max = numbers[0];
        for(int i=0; i< numbers.length; i++)
        {
            if(numbers[i]<min) min = numbers[i];
            if(numbers[i]>max) max = numbers[i];
        }
        Duo pair = new Duo (min, max);
        return pair;
    }
}

private class Duo
{
    public int num1;
    public int num2;

    public Duo(int num1, int num2)
    {
        this.num1 = num1;
        this.num2 = num2;
    }
}

Note that inner class is private.

- Can’t be used elsewhere.
- Indeed, not useful elsewhere, or should have made it its own class.
Another Inner Class Example

```java
public class Square {
    public double sideLength = 0.0;
    ...
    //get area and circumference of inscribed circle
    private ????? getAreaAndCircumference() {
        double radius = sideLength/2.0;
        double area = Math.PI * radius * radius;
        double circumf = 2.0 * Math.PI * radius;
        return area???, circumference????;
    }
}
```

// ok, I admit this is somewhat contrived,
// but you get the idea

**Problem:** Can’t return both the circumference and the area.

**Solutions:**

1. **Put in double array.**
   - problem: awkward!

2. **Make instance variables.**
   - problem: area and circumf of circle don’t belong in Square!!!

3. Make new inner class that holds both.
   - notice the method is private, so not used in other classes.
public class Square {
    public double sideLength = 0.0;

    ...various methods...

    private Circle getAreaAndCircumference() {
        Circle c = new Circle();
        c.radius = sideLength / 2.0;
        c.area = Math.PI * c.radius * c.radius;
        c.circumf = 2.0 * Math.PI * c.radius;
        return c;
    }
}

private class Circle {
    public double area = 0.0;
    public double circumf = 0.0;
    public double radius = 0.0;
}

Note that inner class is private.

- Can’t be used elsewhere.
- Indeed, not useful elsewhere, or should have made it its own class.
public class LinkedList
{
    private Node listNode;

    public int dataAtFirstNode() {...}
    public int dataAtNextNode() {...}
    public int dataAtPreviousNode() {...}

    ...various methods...

    private class Node
    {
        public int data;
        public Node nextNode;
    }
}
Warning! Inner Class Security!

- Inner classes are a security risk.
- JVM (Java Virtual Machine) converts them into separate classes.
  - no longer inner
  - You can even see these classes. Look for OuterClassName$InnerClassName.class in your files.
- You may think they are “hidden”, but they are not!
Packages

- Suppose you have related classes.
  - animals: Pig, Cat, Yak, …
  - instruments: Flute, Drum, Piano, …
  - CS Classes: CS435, CS417, CS427, …

- Would make most sense to group these together.
  - Put similar blueprints in the same place.
  - Put similar classes in folders.
Java packages are folders.

- Recall “import java.io.*;”
  - This means, please get all of the input/output classes.
- What are they?
  - Related classes like File.java, InputStream.java, OutputStream.java, etc.
- Where are they?
  - All grouped together in folder called “io”.
  - And this is in a folder called “java”.
Package Structure

java

io

InputStream.java
import java.lang.Math;
import myFolder.myFolderAlso.MySpecialClass;

class SimpleExample {
    ...
}

These “packages” are just directory structures. So the class “MySpecialClass” is in a folder called “myFolderAlso” which is in a folder called “myFolder”.

This statement means: make MySpecialClass available for use by this (SimpleExample) class.
package cartoon.character;

import java.io.*;
import cartoon.products.acme.Glider;

public class RoadRunner
{
    ...
}

Says: This file lives in the folder (and subfolder) “cartoon\character”.

Your responsibility to save it into those folders.

In Eclipse, select new → package, and it will create the folder for you.

NOTE: package declaration must be first statement in the file.
Your Turn

Based on previous example…

- Create the outline for a Coyote class.
  - Put it in the correct package!

- Ditto for an Anvil class.
  - Put it in the correct package.
Using Your Own Packages

- So now you can **write 50 different classes** … and stuff ‘em wherever you like, **in any folders**.

- To use one, just **import** it

```java
import cartoon.products.acme.Anvil;

public class Dynamite {
    public static void main(String[] args) {
        Anvil heavy = new Anvil();
        heavy.drop();
    }
}
```

Your method, in your Anvil class, that lives in a different folder.
Classpath for Packages

- **Must set CLASSPATH variable on your computer.**
  - Tells Java where to look for your packages.
  - Otherwise compiler can’t find your packages.

- **Called an “environment” variable.**
  - The Java compiler asks the environment variable where to go.

- **Eclipse does this for you!**
  - But if you aren’t using Eclipse, then read on.
Setting the Classpath

1. Type “javac myClass.java –classpath path1; path2…” where path1 and path2 are the directory paths to your packages. Ditto for java.

2. Or type “c:\>set classpath=path1; path2” with no spaces around the equals sign. This sets it for as long as your command prompt window remains open.

3. Or in Windows go to the Control Panel, open System, click on the Environment Variables button (may be under a tab called Advanced). Add a variable called CLASSPATH with the values path1, path2, etc. This sets the classpath at startup so it applies to all windows always (won’t ever have to type it again – until you add a new path!).

Or google how to set your classpath. Zillions of sites with instructions.
For example, on a Windows laptop, I could add the following environment variable.

- CLASSPATH = .;C:;C:\coolStuff

- This says, first look in “.” (the current directory).
- Then look in C:
- Then look in C:\coolStuff
- Each possible folder is separated by a “;”

- Note: I don’t specify the package folders – just specify the path to the folder that contains the top level of the package.
  - e.g., if my code says “package cartoon.character;”, then I specify the classpath to the folder that contains “cartoon”, and no further.
Package Variables/Methods

- Suppose I have a bunch of classes in a Simpson package.

- Suppose one class has this method and variable…

```java
public class Homer {
    int age = 32;
    String noise() {
        return "doh!";
    }
}
```

- What’s missing?
  - The public, private!
Package Scope

Called “package scope”.

- Any other class in the package can access the method or variable!
- Classes outside the package cannot access the method or variable.

So you can have

- public
  - Everyone can use it.
- private
  - Only the class can use it.
- or nothing
  - Only classes in the same package can use it.

Discourage it!

- Dangerous with good hackers.
- As rule, make everything as private as possible.
Static Variables

- Variables that belong to the class, not the object.
  - Class is blueprint for an object.
  - Object only exists after instantiated class.
  - So static variables exist before instantiation!

- Declare as class attributes with
  - public static int populationSize = 1000;
  - public static String name = "Jose";
Why Use Static Variables

1. Handy if you want all objects to have exactly the same feature.
   - e.g., all Persons have same name “Jose”.
   - e.g., all Cars have same color “blue”.

2. Can call without instantiation.
   - String yourName = Person.name;
   - String color = Car.carColor;
   - int cityPopulation = City.populationSize;
public class Cow
{
    public static int numberOfCows = 0;

    public Cow()
    {
        numberOfCows++;
    }

    public String cowSound()
    {
        return "moooooo";
    }
}
public class CowHerd
{
    public static void main(String[] args)
    {
        Cow daisy = new Cow();
        Cow hank = new Cow();
        Cow fred = new Cow();

        System.out.println("I have "+ Cow.numberOfCows + " in my herd");
    }
}

So, what’s this print?
(Hint: each Cow instantiation incremented the counter.)
Login Static Example (With a Constant)

```java
public class Login {
    public static final int MAX_NUMBER_LOGINS = 10;
    private static int numberOfLogins = 0;

    public Login() {
        if (numberOfLogins < MAX_NUMBER_LOGINS) {
            numberOfLogins++;
        } else {
            System.out.println("Too many logins!");
        }
    }

    public String readUserName() {
        String userName = JOptionPane…etc…
    }
    …etc…
}
```

“final” means it’s a constant. NOT a variable. Can’t change it. Usually in uppercase letters.

So this keeps track of the number of Login objects. And complains if too many.
Static Methods

- Methods that belong to the class, not the object.
  - Class is blueprint for an object.
  - Object only exists after instantiated class.
  - So static methods exist before instantiation!

Examples

- String answer = JOptionPane.showMessageDialog(...);
- int x = Math.sqrt(25);
- int y = Math.abs(-32.4);

Declare as

- public static int getPopulationSize() {...}
- public static String getName(...)
Static Methods Call Static

Static methods can

1. Call other static methods.
2. Instantiate new objects and use their methods.
3. But can never call non-static methods without an object!

Why?

- Consider following example.
public class CowHerd {
    public static void main(String[] args) {
        Cow daisy = new Cow();
        Cow hank = new Cow();
        Cow fred = new Cow();
        getHerdSize();
        getHerdLocation();
    }

    public static int getHerdSize() {
        return Cow.numberOfCows; //remember this?
    }

    public void getHerdLocation() {
        ...
    }
}

This works. The static main() can call the static getHerdSize() as well.

This does not work! The main() is static, and we did not bootstrap a CowHerd. The herd’s location doesn’t even make sense until we create a CowHerd object and then tell that object where it is “located”.

Think of it this way.
(1) getHerdLocation() must be called from an object.
(2) The main() was not called from an object (it was static).
(3) So the “non-object” main can’t call the “object-oriented” getHerdLocation().